AD-A100 404 OFFICE OF THE UNDER SECRETARY OF DEFENSE FOR RESEARCH--ETC F/G 9/2
DEPARTMENT OF DEFENSE REQUIREMENTS FOR ADA PROGRAMMING SUPPORT --ETC(U)
FEB 80

UNCLASSIFIED NL

1 OF 1
AD A
100404

END
DATE
FILMED
7-81
DTIC

LEVEL

DEPARTMENT OF DEFENSE

AD A100404

REQUIREMENTS FOR Ada PROGRAMMING SUPPORT ENVIRONMENTS.

DEPARTMENT OF DEFENSE UNITED STATES OF AMERICA

"STONEMAN"

February 1980

81 6 19 003

OFFICE OF THE UNDER SECRETARY OF DEFENSE
WASHINGTON, D.C. 20301

RESEARCH AND
ENGINEERING

Dear Friend of Ada:

Thank you for your interest in Ada.

Your name has been added to the Ada mailing list, and occasionally you will receive information from the Ada Joint Program Office concerning the status of the Ada program.

Under the Freedom of Information Act, the Ada Joint Program Office (AJPO) mailing list is being made available on the USC-ECLB computer. If you object to inclusion of your name on this public list, please inform the AJPO in writing. To help keep the list up-to-date, please notify the AJPO of address changes.

Sincerely,

[illegible]

for Larry E. Druffel, Lt. Col., USAF
Director, Ada Joint Program Office

STONEMAN

February 1980

H

OFFICE OF THE UNDER SECRETARY OF DEFENSE
WASHINGTON, D.C. 20301

RESEARCH AND
ENGINEERING

3D1079 Pentagon
18 February 1980

PREFACE

The U.S. Department of Defense Common High Order Language program was initiated in 1975 with the goal of establishing a single high order language for new DoD embedded computer systems. We are now approaching that goal. A significant reduction in the number of languages approved for new systems was achieved through the issuance of DoD Instruction 5000.31 in November 1976, the technical requirements for the common language were finalized in the Steelman report of June 1978, the preliminary language design was completed and named Ada in June 1979, and extensive test and evaluation of the design has just been completed. The effort thus far has been coordinated through the High Order Language Working Group (HOLWG) of the DoD Management Steering Committee for Embedded Computer Resources.

In the next few months Ada will be added to the DoD list of approved languages, the HOLWG will be replaced by a permanent Ada Language Control Board (ALCB), and national and international standards are planned. The final phase of refinement of the language design will be completed and issued in July 1980. Development of an Ada Compiler Validation Capability (ACVC) was begun in September 1979, and multiply targeted production compilers are planned by the Army and Air Force with work to begin in March and July 1980 respectively.

It was recognized from the beginning that the major benefits to DoD from a common language would be economic and would derive from Ada's appropriateness to military applications, from the portability that comes with a machine independent language, from the availability of software resulting from acceptance of the language for nonmilitary applications, and most importantly from the use of Ada as a mechanism for introducing and distributing effective software development and support environments to those developing and evolving military systems.

The Ada effort is now at a major transition point as the emphasis shifts from the language design to its introduction and use. The Stoneman is the first major accomplishment of this new phase. The Stoneman paints a broad picture of the needs and identifies the relationships of the parts of an integrated Ada Program Support Environment (APSE). It develops a model which reflects an understanding of both the realities of current practice and a realistic appraisal of the possibilities for more effective software development and support environments. It calls for the integration of conventional software tools into a framework that is sufficiently open ended to accommodate a wide variety of programming methodologies and automated software tools currently unavailable or unused in military systems.

The Air Force will act as the lead Service in the development of an integrated software environment for Ada. Their draft RFP has been issued and the responses reviewed. The final RFP will be issued in April 1980 and will call for several parallel efforts for detailed designs of APSEs in compliance with the Stoneman. Widespread review is planned at major milestones in the designs. After a six month initial design effort one or more of the designs will be continued into implementation. The Stoneman will not be superseded until evaluation of the initial designs has been completed. Products are planned for two years from the start of the designs.

At the same time, the Army will be developing a few critical software tools to enable more effective use of Ada in the context of existing software environments. Bids on the Army efforts have been reviewed and the contractor will be announced in February 1980. The Army also will take the lead in Ada related education and training efforts. The latter activities are coordinated through the Ada Education and Training Committee chaired by DARPA.

The Stoneman will also play a major role in the Ada Joint Project Office (AJPO) now being established. This will be a funded office with full time personnel and will be the principal DoD agent for development, support and distribution of generic tools, common libraries, and environments for Ada. The AJPO will continue the coordination of all generic Ada efforts within DoD, including ongoing efforts in the Army, Air Force, DCA, and DARPA, to ensure their compatibility with other Service and DoD Agency requirements, to avoid duplicative efforts and to maximize sharing of resources. The HOLWG will remain the point of contact for Ada activities until the ALCB and AJPO are established.

We are extremely grateful for the laudatory effort of John Buxton in developing the Stoneman. It represents the state-of-the-art in programming environments in the context of Ada and has been developed in as much depth as can be expected without more intensive detailed studies. Initial reviews by some 50 people have been very positive. We are also indebted to Vic Stenning for his able assistance, to the many individuals who have contributed to the various iterations in the Pebbleman/Stoneman process, and to DARPA and the British Ministry of Defense for sponsoring John and Vic's work.

Although this brings the Stoneman series to an end, it is only a beginning for the extensive work that lies ahead. The Ada language provides a mechanism for considerable cost savings and for introducing modern software technology into operational use in military systems. That potential, however, can be achieved only through continual effort and dedication to the task. The Stoneman provides guidance for an important facet of that effort.

David A. Fisher

David A. Fisher

Staff Specialist for Computers, Communications, and Command and Control

HARVARD UNIVERSITY
CENTER FOR RESEARCH IN COMPUTING TECHNOLOGY

18 February 1980

ACKNOWLEDGEMENTS

It is with great pleasure that I acknowledge the very significant part Vic Stenning has played in the preparation of this document by working with me as co-author during the major redraftings.

The work embodied in the Stoneman would not have been possible without the generous cooperation and enthusiastic help of the very many people who have provided substantial criticisms and/or alternative versions for sections of the text at various stages in the project. Particularly substantial contributions have been made by R.F. Brender, T.E. Cheatham. R.A. Converse, L.E. Druffel, R. Firth, H. Fischer, D.A. Fisher, R.L. Glass, H. Hart, D.S. Johnson, M.M. Lehman, L. MacLaren, D. Notkin, I.C. Pyle, D.F. Roberts, P. Santoni, E. Satterthwaite, T.A. Standish, S.L. Squires, J.A. Townley and P. Wegner.

John Buxton

Professor John N. Buxton

AIKEN COMPUTATION LABORATORY
Cambridge, Mass. 02138

1 INTRODUCTION

1.A This document specifies the requirements for an Ada Programming Support Environment (APSE). It provides criteria for assessment and evaluation of APSE designs, and offers guidance for APSE designers and implementers.

1.B The purpose of an APSE is to support the development and maintenance of Ada applications software throughout its life cycle, with particular emphasis on software for embedded computer applications.

1.C The three principal features of an APSE are the data base, the (user and system) interface and the toolset. The data base acts as the central repository for information associated with each project throughout the project life cycle. The interface includes the control language which presents an interface to the user as well as system interfaces to the data base and toolset. The toolset includes tools for program development, maintenance and configuration control supported by an APSE.

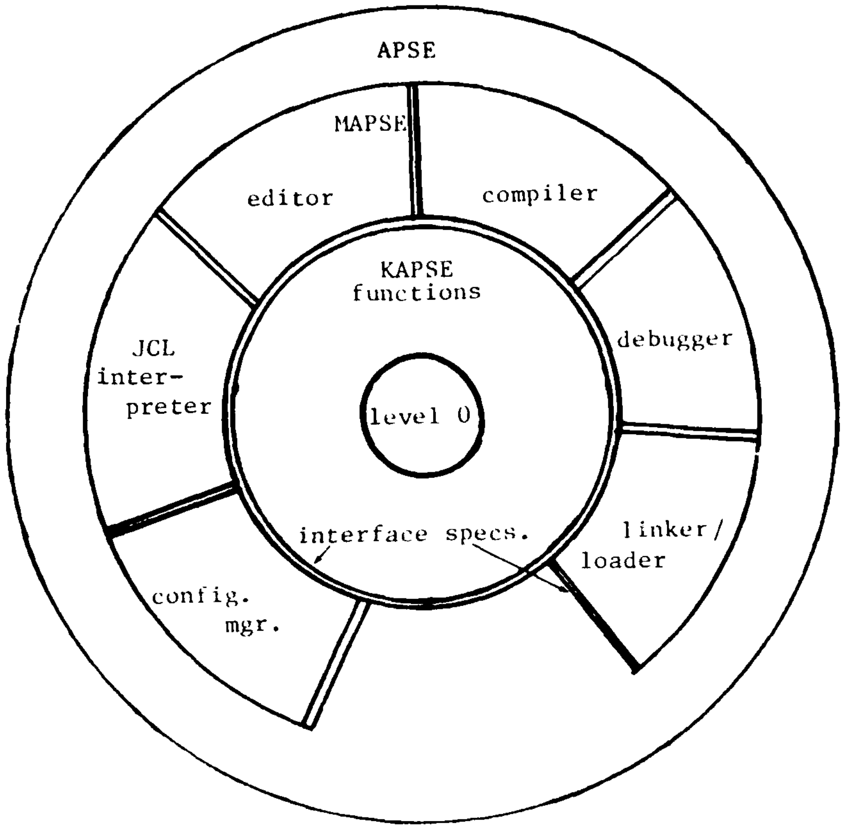
1.D A further goal of great importance in some areas of Ada usage, such as within the DoD, is that of portability both of user programs and of the software tools within the APSE. The Stoneman,therefore, goes on to indicate an approach to portability by giving requirements for two lower levels within the APSE; the Kernel (KAPSE) and the minimal toolset (MAPSE)

1.E It is convenient to represent an APSE which addresses these problems as a structure with a number of layers or levels:

level 0:	Hardware and host software as appropriate
level 1:	Kernel Ada Program Support Environment (KAPSE), which provides database, communication and run-time support functions to enable the execution of an Ada program (including a MAPSE tool) and which presents a machine-independent portability interface.
level 2:	Minimal Ada Program Support Environment (MAPSE) which provides a minimal set of tools, written in Ada and supported by the KAPSE, which are both necessary and sufficient for the development and continuing support of Ada programs.
level 3:	Ada Program Support Environments (APSEs) which are constructed by extensions of the MAPSE to provide

fuller support of particular applications or methodologies.

1.F This structure is illustrated in the diagram below.



1.G Section 2 of this document provides some perspective concerning current practice and the mechanism for making the transition from current systems to the Ada environment. Section 3 presents general principles for the overall design of an APSE system. Section 4 presents requirements for the main individual components of an APSE. Sections 5 and 6 respectively present the requirements for a kernel interface (KAPSE) and minimal toolset (MAPSE). Section 7 presents proposals for further components which are candidates for inclusion in an APSE.

1.H Where appropriate, requirements are supplemented by notes which motivate and explain the requirements. In particular, sections 4, 5 and 6 include sections of notes which respectively supplement the requirements on APSEs, KAPSEs and MAPSEs.

1.J It is possible to take a broader and more general view of programming environments as embodying and supporting the complete integrated process of program design and evolution. This generality is regarded as beyond the present scope of the Stoneman; however, the aim is that the present document should not exclude a more general view being developed and so it is intended to be "upwards compatible" in all critical areas.

2. PERSPECTIVE

2.A CURRENT PRACTICE

2.A.1 According to Department of Defense studies more than half of DOD software costs are associated with embedded computer systems (Fisher, D.A., DOD's common programming language effort, Computer (March 1978), 25-33). As Fisher notes:

> "Embedded computer software often exhibits characteristics that are strikingly different from those of other computer applications.... Many embedded computer applications require software that will continue to operate in the presence of faults.... The applications may require the monitoring of sensors, control of equipment displays or operator input processing. They must interface to special peripheral equipment.... Software must sometimes be able to respond at periodic (real time) intervals, to service interrupts within limited times, and to predict computation times.... In many applications..it is necessary to access, manipulate and display large quantities of data. Much of this data is symbolic or textual rather than numeric and must be organized in an orderly and accessible fashion."

Typically, hardware costs now account for only some 15% of project costs, with 70% to 90% of software costs arising in the long term life cycle maintenance and support phase of the system.

2.A.2 In order to establish the background, we give here a brief summary of typical current practice in many establishments engaged in developing embedded computer systems for military and industrial purposes.

2.A.3 REQUIREMENTS: Particularly in the military field there are formal steps and documentation required at each stage in the system development. However, especially with advanced systems, the initial requirements specification may be imprecise and generalized.

2.A.4 DESIGN: The design stage often involves considerable interaction with the customer and may also be dependent on experimental results from simulation and prototype studies. This can lead to experimental or iterative design techniques.

2.A.5 IMPLEMENTATION: In many cases this is carried out on a host computer, of reasonable size, batch or interactive about equally probable, with mainframe vendor's software support. Much of the programming is done in a high level language, but some is done directly in assembly code. The compiler is viewed primarily as a means to generate and document assembly code and is therefore expected to produce efficient but understandable assembly code. The link-edit phase generates target configuration load tapes.

For some projects there is no host computer and implementation is directly on the target machine with minimal support facilities.

2.A.6 TESTING:

(a) Syntactic and static semantic errors are resolved simply at source code level by the compiler;

(b) The next stage of module testing may be done in the host if an instruction level simulator for the target exists; this is more likely if some simplified environment simulator is available on the host to provide test data input;

(c) The real testing takes place in the "software test lab." This consists of a target configuration, either free-standing or wired up to:

i) A Simulator providing simulated real time inputs; this is programmed on another computer, using random data generation or real mission records.

ii) A hardware operations console, preferably enhanced by a monitor computer to trace data bus traffic and execution of the test runs.

At least 50% of the debugging takes place here, where size and time problems finally have to be resolved. The technique used is that of patching and retesting with later manual updating of the corresponding source texts.

(d) The testing phase of a project is often more expensive than development.

2.A.7 DATA HANDLING: Typically about half the embedded computer memory may contain data, partially fixed and partially mission-dependent, e.g., flight performance details and maps respectively. Although code is rarely classified, data is often classified and can only be loaded in a secure environment.

Data preparation and checking may itself be a substantial task.

2.A.8 ERROR REPORTS: A necessary and continuing task is administering error reports and fixes, together with configuration, version and release control throughout the lifetime of the released system. This is regarded as a data processing task and may be carried out manually or, with larger applications, by a computer data processing system.

2.A.9 CYCLE TIME: A full recompile cycle, with optimising,reconfiguring, etc., can vary in length from 2 days to 2 weeks, depending on size of project, quality of software tools, etc.

2.A.10 MAINTENANCE: This is normally performed by personnel who were not associated with the original software development, often at the customer site. The practice is developing whereby the hardware/software system used during development is contracted as a deliverable and shipped with the product, enabling fault correcting and system upgrading to continue much as described under TESTING above. The system typically continues to undergo change throughout the lifecycle and the annual extent of changes may well be of the same order of magnitude as that of the original system.

2.A.11 CONCLUSIONS: The main features which have a major influence on the activity of developing real time embedded computing systems would seem to be:

a) The evolving nature of the requirements. where an initial general specification is made more precise as the project proceeds. This produces rapid iterations between design and experiment. Typically, in many cases less time is spent in coding than in interacting with the system designers and other component designers such as hardware engineers.

b) The overriding concern with size and time constraints on the target machine. This results in the need for highly optimised code, densely packed data, sophisticated overlay and back-up schemes (and therefore leads to very long complete system recompile times) and on-line debugging.

2.A.12 Current practice is to use few general purpose tools, often limited to a vendor-supplied compiler and linker plus maybe an editor. Various special purpose tools are built in a somewhat project specific way, either by building from scratch or by modifying other tools; these may include overlay optimisers for discs or drums, load tape builders, patch override consolidators, etc.

2.A.13 An environment simulator is of particular importance as a project-specific tool which is always required and may well require as much effort to design and build as a major component of the system.

2.A.14 In past and in much of current practice, the concept of a support system is not much in evidence. The tools available are not well integrated, nor do they form a complete set.

2.A.15 In particular, the crucial problem of long term configuration control is normally addressed by a combination of ad-hoc manual and data processing techniques, and successful configuration control is therefore critically dependent upon management skills.

2.B APSE OVERVIEW

2.B.1 The overall objective of an APSE is to offer cost-effective support to all functions in a project team engaged in the development, maintenance and management of a software project, particularly in the embedded computer system field, throughout the lifetime of the project.

2.B.2 An APSE adopts a host/target approach to software construction. That is, a program which will execute in an embedded target computer is developed on a host computer which offers extensive support facilities. Except where explicitly stated otherwise, this document refers to an APSE system running on a host machine and supporting development of a program for an embedded target machine.

2.B.3 An APSE offers a coordinated and complete set of tools which is applicable at all stages of the system life cycle, from initial requirements specification to long-term maintenance and adaptation to changing requirements.

2.B.4 The tools communicate mainly via the database,which stores all relevant information concerning a project throughout its life cycle. The database is structured so that relationships between objects in the database can be maintained, in order that configuration control problems can be resolved.

2.B.5 Individual functions supported by the tools in an APSE include:

(1) Creation

It is possible to create database objects which contain specifications, design documentation, program source text, program documentation,test data, and so on.

(2) Modification

A database object can be modified to produce a new object (or a new version of the same object), for example, by editing.

(3) Analysis

The entities in a database object can be analyzed, producing a new object which records the results of this analysis. Examples of such analysis are set/use and cross reference listings.

(4) Transformation

The representation of a database object may be changed by transformation tools. These may include optimisers which optimise a program object with respect to some constraint, parsers and code generators.

(5) Display

Objects can be displayed on terminals, printers, and so on.

(6) Linking

A collection of compiled code objects can be consolidated, resulting in a new object ready for loading and execution.

(7) Execution

Once a program has been compiled and linked, it can be loaded and executed, possibly with an appropriate environment being used to supply test information and to monitor execution.

(8) Maintenance

The APSE must enable configuration control to be maintained. For any configuration of software, it is necessary to be able to determine the origin and purpose of each component of the configuration and to control the process of further development and maintenance of the configuration.

2.B.6 The user interface offered by an APSE is independent of the host machine.

2.B.7 At all stages of the development of a program--design, coding, testing, maintenance--an APSE encourages the programmer to work in Ada source terms, rather than in terms of the assembly language of the particular host or target machine.

2.B.8 Extension of an APSE toolset requires knowledge only of the particular APSE and of the Ada language. A new tool--for example, an environment simulator--is written within the APSE. This tool can then be installed as part of the APSE and subsequently invoked.

2.B.9 An APSE supports the use of libraries of standard routines for incorporation in programs written for both host and target machines.

2.B.10 The above paragraphs outline the facilities offered by an APSE to its users in support of Ada programming. However, a further requirement is for portability both of APSE tools between, for example, APSEs hosted on different machines and of complete APSE toolsets. To address this aim and to indicate a means of implementation of an APSE designed to provide portability, this document gives requirements for a low level portability interface and support function set (the KAPSE) together with a minimal toolset (the MAPSE).

2.B.11 The purpose of the KAPSE is to allow portable tools to be produced and to support a basic machine-independent user interface to an APSE. Essentially, the KAPSE is a virtual support environment (or a "virtual machine") for Ada programs, including tools written in Ada.

2.B.12 The declarations which are made visible by the KAPSE are given in one or more Ada package specifications. These specifications will include declarations of the primitive operations that are available to any tool in an APSE. They will also include declarations of abstract data types which will be common to all APSEs, including the data types which feature in the interface specifications for the various stages of compilation and execution of a program.

2.B.13 While the external specifications for the KAPSE will be fixed, the associated bodies may vary from one implementation to another. In general all software above the level of the KAPSE will be written in Ada, but the KAPSE itself will be implemented in Ada or by other techniques, making use of local operating systems, filing systems or database systems as appropriate.

2.B.14 The minimal APSE (MAPSE) is one which provides a minimal but useful Ada programming environment and supports its own extension with new tools written in Ada. Hence, the MAPSE is an APSE and must meet the general requirements set down for APSEs.

2.B.15 For many important activities during a project life cycle as listed below, the only support offered by the MAPSE consists of general text manipulation facilities. A more comprehensive APSE will offer specialized tools to support a wide range of these activities, possibly including:

1) Requirements Specification
2) Overall System Design
3) Program Design
4) Program Verification
5) Project Management

2.B.16 Clearly, the MAPSE does not emphasize any particular development methodology at the expense of any other. However a comprehensive APSE may encourage, or even enforce, one specific system development methodology.

2.C STRATEGY FOR ADVANCEMENT

2.C.1 It is to be expected that many systems which offer the use of Ada as a programming language will come into existence without full, or indeed any, regard to the requirements of Stoneman. Translators will be implemented, for example, within existing support environments hosted on existing operating systems, together with existing software tools and techniques and possibly with the implementation of some further Ada-related tools.

2.C.2 In cases where there is a large current investment in software projects, written originally in other languages and for which long term maintenance must be continued and improved, a viable policy may be to implant APSE-built tools into the existing environment or toolset with a view to improving the existing environment for maintenance. This could provide valuable technology transfer at the environment level rather than the language level.

2.C.3 In other cases, Ada support environments will be constructed offering the use of the language together with a range of facilities which differ markedly in content and/or structure from those proposed here . For example, highly-integrated top-down development systems may be produced in some programming research establishments , meeting the APSE requirements but not reflecting the KAPSE/MAPSE structure.

2.C.4 All such systems,of course,represent entirely valid approaches to use of the Ada language. A further intent of the Stoneman, however, is to propose a way forward towards the goal of portability. APSEs will address this requirement where relevant by making explicit a KAPSE portability level.

2.C.5 In order to achieve the long-term goal of portability of software tools and application systems dependent on them, it is intended that conventions and, eventually, standards be developed at the KAPSE interface level. This level will then serve as specifying a portability interface and separate tools or integrated sets of tools which meet the KAPSE interface requirements will be readily portable between hosts.

2.C.6 At this time, no attempt is made to specify in the Stoneman a long-term standard set of KAPSE interfaces, on the grounds that such standardization would be premature. The document gives requirements for APSEs in general and for KAPSEs and MAPSEs in particular. In later Appendices, provisional examples will be given of specifications for some KAPSE interfaces in the form of Ada package definitions. It is intended that such Appendices be separately distributed as available.

2.C.7 Progress towards the long-term goal of a wide measure of portability is expected to be achieved by a process of competitive design and evaluation of APSEs and their associated KAPSEs.

2.C.8 It is expected that in response to Stoneman and other initiatives designs for MAPSEs will be put forward in the immediate future for consideration by DOD components and others. These alternative designs will be considered and evaluated and may lead to iterative changes in the requirements.

2.C.9 Furthermore, it is expected that one such MAPSE, and more particularly its basic kernel or KAPSE, will achieve a sufficiently wide measure of acceptance for it to become a de facto convention and, eventually, for the KAPSE specifications to be considered for standardization within the DOD.

2.C.10 Individual tools or sets of mutually dependent tools implemented on the DOD-KAPSE would then become fully portable within the DOD and other applications areas with DOD-compatible systems.

3. GENERAL GUIDELINES

The principles listed here apply in general to all aspects of the support system and so they are presented in a relatively unstructured way. They serve as design guidelines and provide criteria of choice for design decisions.

3.A SCOPE: An APSE shall provide a program development and maintenance environment for embedded computer system projects involving Ada programs, with the intent of improving long-term cost effective software reliability.

3.B QUALITY: An APSE shall reflect the priorities for software quality in military embedded computer applications; that is, reliability, performance, evolution, maintenance and responsiveness to changing requirements.

3.C SIMPLICITY: The structure of an APSE shall be based on simple overall concepts which are straightforward to understand and use and few in number. Wherever possible, the concepts of the Ada language will be used in the APSE.

3.D LIFE CYCLE SUPPORT: Support shall be provided to projects throughout the software life cycle from requirements and design through implementation to long term maintenance and modifications.

3.E PROJECT TEAM SUPPORT: An APSE shall support all functions required by a project team. These functions include project mangement control, documentation and recording, and long-term configuration and release control.

3.F USER HELPFULNESS: High priority will be given to human engineering requirements in the design. The system shall provide a helpful user interface that is easy to learn and use, with adequate response times for interactive users and turn-round times for batch users.

3.G UNIFORMITY OF PROTOCOL: Communications between users and tools shall be according to uniform protocol conventions.

3.H SYSTEM PORTABILITY: An APSE shall be portable so far as practicable. This will normally be achieved by writing the system in Ada, and by following the KAPSE design model as required in this document.

3.J PROJECT PORTABILITY: An APSE shall be designed to facilitate the easy movement of project support from one host machine to another.

3.K HARDWARE: an APSE will be designed to exploit, but not demand, modern high capacity and high performance host system hardware.

3.L ROBUSTNESS: An APSE will be a highly robust system that can protect itself from user and system errors, that can recover from unforseen situations and that can provide meaningful diagnostic information to its users.

3.M INTEGRATED: An APSE shall provide a well-coordinated set of useful tools, with uniform inter-tool interfaces and with communication through a common database which acts as the information source and product repository for all tools.

3.N GRANULARITY: Tools will be designed where appropriate to have separable limited function components that are composable, user selectable and communicate through the common data base.

3.P OPEN-ENDED: An APSE shall facilitate the development and integration of new tools. It shall permit improvements, updates and replacement of tools.

4. REQUIREMENTS FOR APSEs

4.A APSE DATABASE REQUIREMENTS

4.A.1 The database is the central feature of an APSE system. It will act as the repository for all information associated with each project throughout the project life cycle.

4.A.2 The database shall offer flexible storage facilities to all APSE tools.

4.A.3 A separately identifiable collection of information in the database in known in this document as an "object". Every object stored in the database is accessed by the use of its distinct name. The database shall permit relationships to be maintained between objects.

4.A.4 The database shall permit the user to designate several distinct database objects as forming a "version group". The user shall be permitted to designate one object within the group as being the preferred (or default) version. A method of access shall be offered in which an incomplete object name is provided, sufficient to identify explicitly a version group but not one object within that group; with such access, the preferred version is selected. Every object within a version group shall always be accessible by providing the complete object name.

4.A.5 The database shall support the generation and control of configuration objects; that is, objects which are themselves groupings of other objects in the database. The configuration control facilities shall allow access to the objects in a version group by the use of an incomplete name.

4.A.6 Mechanisms shall be provided in the database whereby all database objects needed to recreate a specified object will continue to be maintained in the database as long as the specified object itself remains in the database.

4.A.7 It shall be possible to establish partitions of the information in the database such that, for example, all objects connected with a specific project area can be grouped in a partition. It shall be possible to associate general access controls with partitions.

4.A.8 The database shall support the storage of Ada libraries in source form, and may also support a form where the library object has been pre-compiled for the host or a particular target machine. Facilities for determining the availability and functional specification of library objects shall be provided.

4.A.9 It shall be possible to associate access controls with any object in the database, Such access controls shall be appropriate for the environment in which the particular APSE system is deployed, and shall be commensurate with the requirement that an APSE supports all roles in a project team throughout the lifetime of a project.

4.A.10 The database shall store information which allows management reports to be generated, as required at the particular APSE system.

4.A.11 The capabilities of the APSE database system shall be such that the users may work within the APSE to achieve reliable storage of objects, including long-term storage of archived objects.

4.A.12 The database shall preserve the consistency of the information and relationships it contains.

4.B APSE DATABASE NOTES

4.B.1 OBJECTS: A separately identifiable collection of information in the database is known as an object in this document. An object has a name by which it may be uniquely identified in the database, it has attribucles and it contains information. Typically, an object may contain a separately compilable Ada program unit, a fragment of Ada text, a separable definition, a file of test data, a project requirements specification, an aggregation of other object-names (i.e., a configuration; see below), a documentation file, etc.

4.B.2 VERSIONS: All objects in the database are uniquely identifiable; however, a group of objects may exist as related versions which all may meet the same or closely related external specifications and may therefore be regarded as different versions of the same "abstract object". Within such a group, the user may specify that one object is the normal, default or preferred version which is to be used whenever the user does not indicate a specific one. Typically, in many current systems the concept of the most recent version of a "module" plays an important role and it may be that this methodological choice will be made in many APSEs; however, the requirements do not prescribe this approach.

4.B.3 CONFIGURATIONS: Different collections of objects in a project may be brought together to form different groupings or "software configurations." The differences arise in response to, for example, differing categories of user requirement or differences between peripheral devices on various target systems. Some configurations are long-lived, such as major system releases, and others may be temporary test-beds for development purposes. The relationships between objects in different configurations are in general complex, partially overlapping and not well-structured. Some configurations are related in time, such as consecutive "releases;" others co-exist in time as separate "models". Note that configurations are themselves objects and may therefore exist in version groups.

The system must contain tools to enable the generation, release and subsequent control of a project which exists in multiple configurations.

It is generally necessary to be able to determine for any configuration exactly what are the components of that configuration and to be able to reconstruct in detail the history and antecedents of each component.

The automatic rederivation of configurations as a result of constituent object changes may be a methodological choice in some APSEs.

4.B.4 HISTORY PRESERVATION: It is a requirement of an APSE that configuration control be provided. In general this necessitates recording and preserving sufficient information to establish, for any extant configuration, its precise constituents and all relevant information to support their repair or modification.

It is therefore a requirement at the KAPSE level that history attributes be maintained for all objects (see 5.A.5 below) as a basis for a configuration control system. At the MAPSE level a configuration control system is required but not specified in detail.

The detailed requirements on configuration control systems are left to some extent open to design choice. One position to take is that no object whatever can be deleted from the database if it is referenced in the history attribute of any other object. This maximises reliability and maintainability and in many application areas, if combined with an effective archiving system, would be the preferable approach.

However, in other areas the requirements may differ and may indicate that indefinite preservation should be the privileoge of specified objects only (see 4.A.6) and objects not so specified,under managerial control, may be purged from the database.

4.B.5 PARTITIONS: The partition level is the highest level grouping in the database. It exists primarily for managerial purposes as a means of applying broad access and bugetary control to large collections of information associated with projects. It can also play a part in implementation strategies designed to improve access to specifically important partitions in large data bases.

4.B.6 ACCESS CONTROLS: The access control requirements on APSEs are expected to be highly specific to the applications area and methodology of each APSE. Some areas will require very detailed controls whereas others will require only a few broad classes of protection.

The key requirement for a KAPSE is therefore that the primitive protection facilities it offers shall be sufficiently general purpose so that it can provide the basis for any required access control system in the APSEs built on that KAPSE.

In order to meet these requirements, the KAPSE must have knowledge of individual user identifications. These may well be handled by the local underlying operating system.

4.B.7 MANAGEMENT REPORTS: The style and content of project management reporting is specific to the project environment and methodology. In general , however, the database should contain information enabling at least two classes of reports to be produced:

(a) Progress reporting--budgets, schedules, review and implementation dates, responsibilities, error report tracing, etc.

(b) Statistical reporting--usage frequencies, system loading, etc.

4.B.8 RELIABILITY: The degree of reliability required in the database is specific to the individual application area as it depends on the economically justifiable level of back-up required on the equipment for the project.

4.C APSE CONTROL REQUIREMENTS

4.C.1 A virtual interface which is independent of any host machine shall be provided for APSE communication.

4.C.2 The virtual interface shall be based on simple overall concepts which are straightforward to understand and use and which are few in number.

4.C.3 The virtual interface shall permit the invocation of individual tools from the APSE toolset.

4.C.4 The user may access the virtual interface from a variety of physical terminal devices.

4.C.5 The virtual interface shall permit the user to interact with the invoked tool and to exercise control over the tool.

4.C.6 APSE tools may access the virtual interface; e.g., to invoke other tools.

4.C.7 An APSE must prevent access from the user which might affect the integrity of the KAPSE and its facilities.

4.C.8 It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set.

4.C.9 Initial user connection to the APSE may require use of the host operating system. A mechanism for returning to the underlying operating system shall be provided in the APSE.

4.D APSE CONTROL NOTES

4.D.1 VARIETY OF CONTROL DEVICES: The virtual interface will in practice, be accessible to the user in so far as practicable from the terminal devices available with a particular system. In general these may be from three categories:

(a) batch terminals

(b) keyboard interactive terminals

(c) high band-width graphics interactive terminals

Where meaningful, the same control signals will be accepted by the terminal interface routines from all devices of these types.

4.D.2 USER INTERACTION WITH TOOLS: The degree of interaction possible between a user and a tool depends on the "granularity" of that style of device; for example, interaction from a batch terminal is limited to initiation of a job, provision of data and parameters and notification of the completion of the job together with its results.

4.D.3 ACHIEVABILITY OF COMMAND FUNCTIONS FROM WITHIN PROGRAMS: The requirement of 4.C.6 is fundamental to the composition of tools.

4.D.4 COMMAND LANGUAGES: The requirements of 4.C.3 and 4.C.4 may well be implemented by a command language (or job control language).

Regardless of the choice of command language, the environment must provide a primitive operation which enables the initiation of a program to be carried out. More precisely, this operation permits a data structure (such as a compiler output) to be executed as a program on the host.

Given this primitive, one possible approach to the implementation of a command language is to use a basic Ada-like language whose facilities, offered by a simple interpreter tool, provide little more than the ability to perform simple editing of command lines and to initiate programs.

The requirement in 4.C.6 indicates that the primitive initiation facility used by the command language will be made available as a library procedure to Ada programs. This will enable the user to construct job control sequences as Ada program texts which initiate other programs. This use may well be subject to some restrictions; for example, to prevent recursive initiation in unsuitable cases.

A more general approach is to regard the user interaction as being expressed entirely within Ada program segments which are executed or interpreted as necessary in the context of relevant points in the APSE database, thus providing a total Ada environment similar, for example, to an Interlisp environment.

In view of this range of possibilities, the detailed choice of command language is left as a design decision for specific APSEs.

4.E APSE TOOLSET REQUIREMENTS

4.E.1 The tools in an APSE shall support the development of programs in the Ada language as defined by the Ada reference manual. In particular an APSE shall support the separate compilation features of the language.

4.E.2 Tools in an APSE shall be designed to meet clear functional needs and shall be composable with other tools in order to carry out more complex functions where appropriate.

4.E.3 Tools shall be written in Ada and where possible shall conform to standard interface specifications.

4.E.4 The set of tools in an APSE shall remain open-ended; it shall always be possible to add new tools.

4.E.5 The communications between tools shall be simple and uniform throughout an APSE toolset. Inter-tool communication shall be via the virtual interface.

4.E.6 The principles for communication between tools and the user shall be simple and uniform throughout an APSE toolset. The uniform principles shall apply to error handling as well as to normal operation of a tool.

4.E.7 An APSE toolset shall offer comprehensive "help" facilities to APSE users.

4.E.8 An APSE toolset shall support its own extension with new tools written in Ada.

4.E.9 An APSE toolset shall permit testing and debugging of any Ada program which does not use machine-dependent features of the language.

It shall be possible to perform such testing and debugging purely in terms of the Ada source text and Ada language concepts (i.e., without reference to the instruction set or architecture of any machine).

4.E.10 An APSE shall permit testing and debugging of an Ada program executing in any target machine supported by the APSE. It shall be permitted for such a program to use the machine-dependent features of the language. The facilities for testing and debugging of target-resident programs should be based upon the equivalent facilities for host-resident programs.

4.E.11 APSE tools shall provide appropriate summary data for management reports and control. Such summary data shall be stored in the APSE database and will be project-dependent in nature.

4.F APSE TOOLSET NOTES

4.F.1 INTER-TOOL COMMUNICATION. Note that where necessary, tools will store information in the database for later use by other tools.

4.F.2 TARGET ENVIRONMENTS. As stated in various sections above, the model of program development expressed in the APSE approach is that of a host-target system where the host offers the vast majority of the support facilities. The whole purpose of the APSE, however, is to develop and support target machine programs and, in embedded computer systems in particular, final testing on the target machine is normally essential. The intention is for that testing to be carried out in Ada terms so far as practicable.

Four general styles of target resident testing are envisaged:

(a) Down-line testing via a host-target machine link with a target test supervisor resident in the target. The APSE is regarded as distributed between the machines, and the target test supervisor is part of the APSE.

(b) Remote testing where the target machine is not directly linked to the host but where the target configuration can support APSE-compatible tools to provide a target-resident test environment. The target-resident part of the APSE may be regarded as linked to the rest of the APSE by batch-style communication.

(c) Isolated testing on the target machine in cases where target configuration limitations or the application environment preclude the availability of APSE-compatible facilities on the target. In these cases, testing methods will continue as at present to be specific to the target and the application.

(d) Direct testing in situations where the target machine is the same as the host machine.

In describing an APSE, it is therefore necessary to specify three parameters:

(a) On what host machine does it reside? (e.g., the CDC6600)

(b) What targets does it support in the sense of (a) or (b) above? (e.g., PDP/11)

(c) For which further targets does it generate code? (e.g., Intel 8080)

It is also necessary to specify which tools are appropriate for use on a target in full or possibly in degraded form.

5 REQUIREMENTS FOR KAPSEs

5.A KAPSE DATABASE REQUIREMENTS

5.A.1 Each object in the database shall have a unique name constructed from a sequence of identifiers. Each component of the name shall conform to the Ada syntax for identifiers. Where an object is a member of a version group, a version qualifier may be appended to the name.

5.A.2 The database shall not impose restrictions on the format of information stored in an object.

5.A.3 The database shall permit relationships between objects to be recorded.

5.A.4 Objects in the database shall have attributes.

5.A.5 Every object shall have a history attribute. The history attribute records the manner in which the object was produced and all information which was relevant in the production of the object. The history attributes shall contain sufficient information to provide a basis for comprehensive configuration control. Any necessary constraints shall be imposed on database operations so that the validity and consistency of history attributes is ensured.

5 A.6 Every object shall have a categorization attribute which indicates the category of information contained in the object. It shall be possible for the categorization attribute to be used in such a way that APSE tools are offered protection against accessing an object in a way that is not meaningful (i.e., incompatible with the format and/or content of the object) but are not prevented from accessing an object in any way that is meaningful.

5.A.7 Every object shall have an attribute which indicates access rights to the object.

5.A.8 The database interface shall permit provision of an archiving facility whereby files may be relegated to backing storage media while nevertheless retaining the integrity, consistency, and eventual availability of all information in the database.

5.A.9 The database shall allow APSE tools to access both the information content of objects and the attributes of objects, and to traverse the networks formed by relationships between objects. Access protection shall be applied to attributes so that attribute consistency is maintained.

5.A.10 It shall be possible for the actual reading and writing of database objects to be performed from within an Ada tool using the standard input/output facilities of the language, as defined in package INPUT OUTPUT.

5.B KAPSE DATABASE NOTES

5.B.1 ATTRIBUTES: An object in the database consists essentially of:

(a) its content; that is the raw information it contains, and

(b) its attributes; that is, meta-information describing the nature of the object, its history,categorization and so on.

In many systems a complete object is represented by associating the attribute information with the directory entry for the object and storing the object itself as an unstructured file.

The attributes which record history, categorization and access rights are mandatory. The list of possible attributes is open-ended and some APSEs may provide further attributes.

5.B.2 CATEGORIZATIONS: Every object has a category attribute. In general, a tool will only access objects of an appropriate category. Other tools, however, may need to access objects regardless of their category; one such tool is a general copying tool. The requirement on the KAPSE encompasses both of these styles of access. However, the requirement does not dictate the manner in which protection is offered, nor that the protection mechanism must actually be implemented within the KAPSE.

5.C KAPSE FUNCTION REQUIREMENTS

5.C.1 The KAPSE shall offer the basic run-time support facilities that are required by Ada programs that execute within the APSE.

5.C.2 The KAPSE shall offer the input/output support facilities that are required by Ada programs within the APSE which use the standard input/output facilities of the language, as defined by package INPUT_OUTPUT.

5.C.3 The KAPSE shall provide the database access functions that are requried by Ada programs within the APSE. This shall include the provision of the primitive functions necessary to permit the implementation of access control and security mechanisms as appropriate.

5.C.4 The KAPSE shall provide a mechanism whereby it shall be possible for one APSE tool to invoke another APSE tool and supply the invoked tool with parameters.

5.C.5 Input/Output support offered by the KAPSE shall be such that package INPUT_OUTPUT can be used by an Ada tool for communication with the control device from which the tool was invoked.

5.C.6 The KAPSE shall provide mechanisms where appropriate whereby asynchronous commands issued by a user at an interactive terminal can be applied to the executing tool.

5.D KAPSE FUNCTION NOTES

5.D.1 INTERACTIVE TERMINAL CONTROL: During execution of a program in an interactive system the requirement exists for the user to be able to have various levels of asynchronous interaction with the program, ranging from requests to terminate irrelevant output to commands to terminate the entire session. A provisional list of functions required is given below.

5.D.2 CONTROL FUNCTION LIST

The KAPSE shall define a fixed set of terminal interface control functions. The list should be short, functionally adequate and human engineered to fit the needs of the terminal user. The following list is proposed:

(a) Issue a request to terminate the current function

(b) Issue a request to terminate the current program

(c) Suspend the current program and establish a new invocation of the command language interpreter

(d) Terminate the current command language interpreter invocation and resume the program suspended when the current CLI was invoked.

(e) Abort the current program and return to its invoker

(f) Abort the current program and return to the nearest command language interpreter level.

5.D.3 CONTROL KEYS: The list of standard control keys used to initiate these interactions is a convention and is parameterised within the KAPSE. The list may be changed if this is necessary to avoid conflict with other local conventions.

5.D.4 PROVISION OF FUNCTIONS: In general, the KAPSE will be specified as a series of Ada package definitions making extensive use of the concept of abstract data types. This technique can be used both to provide KAPSE functions as listed in 5.C. above and to provide data structure descriptions of interfaces as in 5.E. below.

5.E KAPSE INTERFACE REQUIREMENTS

5.E.1 The KAPSE shall implement interface definitions which shall be available to APSE tools. Such interface definitions shall be given in the form of package specifications in the Ada language.

5.E.2 Interface definitions provided by the KAPSE shall encompass

(i) the primitive operations that the KAPSE makes available to APSE tools; these include any operations that may be necessary to supplement the facilities of package INPUT_OUTPUT (see 5.A.10) in order to allow an APSE tool access to all the functional capabilities of the database,

(ii) the abstract data types (type declarations plus operations) that are required to interface the various stages of compilation; these include the data types that are produced by a compilation stage for later use by analysis, testing, or debugging tools.

Certain of the abstract data types of (ii) are considered individually in the following paragraphs.

5.E.3 The source representation of a compilation unit shall be as defined in the Ada reference manual.

5.E.4 An abstract syntax definition of a compilation unit shall be specified.

5.E.5 A post syntactic/semantic analysis intermediate language definition of a compilation unit shall be specified.

5.E.6 An abstract data type definition of an executing Ada program shall be specified. The abstract data type shall offer read/write access to both the code segments and the data spaces of the executing program. When used in conjunction with the full symbol tables (see 5.E.7 below), this abstract data type shall allow the production of source-level debugging tools.

5.E.7 An abstract data type definition of a comprehensive symbol table for a compilation unit shall be specified. In addition to the basic symbol declaration entries, the abstract data type shall encompass source line locations, symbol usage, program topology, and any other information required by basic analysis, testing, or debugging tools.

5.E.8 An abstract data type definition of a "library file" (Ada Reference Manual, section 10.4) shall be specified. This abstract data type shall allow new compilation units to be added to the library file, and shall allow the relationship between compilation units in the library file to be determined.

5.E.9 When used in combination, the symbol table and library file abstract data types shall permit construction of comprehensive symbol table(s) for the (possibly incomplete) program(s) represented by the contents of the library file.

5.F KAPSE INTERFACE NOTES

5.F.1 *INTERMEDIATE REPRESENTATIONS*

In some translation systems there is a clearly defined intermediate language representation level between the syntactic and semantic analysis phases and the code generation phase. This separation is the basis of the retargetable compiler approach whereby several code generators are provided for different targets. It is expected that retargetable Ada translators will be produced and therefore there is a requirement for agreement on this level of representation.

Currently one candidate for an Ada-oriented intermediate representation is available for comment. This is TCOL-Ada and it is described in

> "TCOL-Ada: Revised Report on an Intermediate Representation for the DOD Standard Programming Language." This report is produced by the Department of Computer Science, Carnegie-Mellon University.

An alternative approach to this problem might be based on consideration of other well established machine-independent low level coding schemes such as O-code, (BCPL); P-code, (Pascal); JOCIT, J73/I, J73 JOCIT, (JOVIAL); etc.

The existence of retargetable and separable-phase translator systems does not of course preclude the existence of more traditional compiler designs in which an intermediate code level may well not be distinguishable.

5.F.2 LIBRARY FILES

A library file as defined in Ada groups together all those compilation units (represented as objects in the database) which are inter-related and which eventually will be linked into one or more complete programs.

The technique of implementation of the library file, and hence of separate translation as defined in Ada, is of course a design decision. An example of such a design, funded by the U.K. Ministry of Defense is given in; "Ada Support System Study: Phase 2 and Phase 3 Reports."

5.F.3 APPENDICES

In addition to the examples referenced in 5.F.1 and 5.F.2 work is proceeding on preparing initial examples of Ada package specifications in response to requirements 5.E.6, 5.E.7, and 5.E.8. It is intended that the examples be circulated when available as Appendices to this document.

A candidate for the abstract syntax specification required in 5.E.4 is expected to emerge from current work in the Ada design group.

6. REQUIREMENTS FOR MAPSES

6.A MAPSE TOOLSET REQUIREMENTS

6.A.1 TEXT EDITOR: A standard text editor shall be provided with facilities suitable for editing general text, including specifications, design and other documents,and source programs. The location of text may be by line number, by context or equivalent. The editor shall provide the following functions: find, alter, insert, delete, format, input, output, move,copy and substitute.

6.A.2 PRETTYPRINTER: A display tool is required to format and output textual material ranging from documentation to source programs. More specifically, it shall print database objects in legible formats which depend on the object categorization.

6.A,3 TRANSLATOR: A MAPSE will include at least one Ada translator, which translates source Ada programs into target code for the host and at least one target. The detailed requirements on the translator are that it should interface to the KAPSE as specified in section 5E above and thereby be cooperative with the other tools.

6.A.4 LINKERS: Linkers are required in a MAPSE for both the host and target machines. The facilities needed are:

Partial linking of program units in conformance with language specifications

Creation of an executable program from program units (perhaps partially linked) with the following options:

- Logical to physical mapping
- Overlay management
- Omission of unused compilation units
- Creation of a single load file
- Linkage map including variable types and unit cross reference
- Elimination of redundant generic code bodies.

6.A.5 LOADERS: Off-line and/or down-line loading shall be supported.

6.A.6 SET-USE STATIC ANALYSER: A tool is required to provide a cross-reference map indicating where each data item is changed in value and where it is merely referenced.

6.A.7 CONTROL FLOW STATIC ANALYZER: This tool produces a chart of the program control topology. This will indicate which routines are called from where in the program and may indicate exception scopes and inter-task communication calls.

6.A.8 DYNAMIC ANALYSIS TOOL: On systems with an interactive capability this tool shall provide the following functions:

(a) Snap shot;

(b) Break (with facilities to alter values of variables for interactive use);

(c) Trace;

(d) Interface simulator (for dummy program units);

(e) Statement execution monitor.

(f) Timing analysis.

Where appropriate some or all of these functions shall be available for batch and/or target environment testing.

6.A.9 TERMINAL INTERFACE ROUTINES: Corresponding device handlers shall be provided for each variety of terminal device available on a specific configuration. These interface between the terminal and the relevant functions and data structures in the KAPSE.

6.A.10 FILE ADMINISTRATOR: A file transfer and compare facility shall be provided. A standard transfer format shall be adopted and the following facilities provided:

File Comparison

Error Control

File Transmission

Title Transmission

History Attribute Transmission.

6.A.11 COMMAND INTERPRETER: A command interpreter capable of invoking all APSE tools (potentially with parameters) must be provided by a MAPSE. Users communicating with the command interpreter through an interactive device must be provided with:

1) Facilities for editing and inspecting command lines prior to carrying out the command.
2) Positive responses to all interactive operations.

The command language accepted by the intepreter must enable the storage of sequences of commands in the database for later execution.

6.A.12 CONFIGURATION MANAGER: A tool is required to assist in long term configuration control of projects. As minimal functions this tool will enable interrogation of history attributes and will offer managerial control over the persistence of objects in the database.

6.B MAPSE TOOLSET NOTES

6.B.1 The most common action in a programming environment is the manipulation of objects. In a complex APSE, much of this manipulation is done automatically by many of its advanced tools. In a MAPSE, however, these advanced tools are missing and much of the manipulation must be done manually. The easiest way to ensure that general objects can be manually manipulated is to provide a general text editor that can manipulate objects at a low level.

6.B.2 Many of the objects stored in the database are not structured in ways that are immediately legible. The information in these objects (as well as their attributes), however, must often be studied by users; hence the need for a prettyprinting tool (or set of tools) that transforms database objects into legible and understandable formats. The prettyprinter may perform transformations that include:

1) Binary to ASCII conversion
2) Indentation of Ada programs
3) Formatting of objects that contain linked lists
4) Formatting of history attributes

6.B.3 In order to execute Ada programs (including the MAPSE tools themselves), it is necessary to translate the programs from (high-level) Ada to a (lower-level) executable representation. It is of little importance from the development standpoint (although it may be important from other standpoints) whether the translator that is provided is a compiler, an interpreter, or any other type of translator.

6.B.4 Debugging and testing Ada programs at a low level representation (machine or assembly language level, for instance) defeats much of the purpose of programming in Ada in the first place. Therefore, these facilities must be provided at the Ada source level in a MAPSE. This may require KAPSE level support.

6.C MAPSE LIBRARY REQUIREMENTS

6.C.1 One or more high level I/O packages are required for the host, to extend or to provide alternatives for the package specified in the language manual. Such packages will provide calling conventions and implementations for standard device handling routines. Both host and target resident packages are required as appropriate.

6.C.2 A high-level I/O package is required for each target machine.

6.C.3 A physical file handling package is required for each target, if appropriate.

6.C.4 A file directory system is required for each target.

6.D MAPSE LIBRARY NOTES

6.D.1 The requirements in 6.C.3 and 4 are not very specific as the nature of the target environment is project-dependent. However, some file handling package is required in the target to implement the access of target files by the host and the exchange of files with the host. These requirements are expressed separately; it may be on a specific target that either the local operating system file directory structure, or physical file handlers, or both will be utilized to meet the requirements

6.D.2 Clearly these proposals for MAPSE libraries represent a minimal subset only.

7 APSE COMPONENTS

7.A APSE TOOLS

7.A.1 This section describes tools that fit the APSE requirement but go beyond the MAPSE.

7.A.2 ADA PROGRAM EDITOR: An APSE may provide an editor specifically for Ada source programs. This will assist in the preparation of syntactically correct programs by providing templates for language constructs and by checking input for consistency as appropriate.

7.A.3 DOCUMENTATION SYSTEM: A substantial part of the effort of developing a system is devoted to its associated documentation; for example in specifications, design documents, user manuals and educational material, and so on. An APSE may provide a tool to support the production and control of documentation for a project. This may interface with the editor and with a word processing system, if one is available, together with display tools (such as the prettyprinter required in a MAPSE). It may control graphic displays, figures and text in a unified system. It may be necessary for this tool to be sensitive to the security classification of materials and to implement military documentation standards where required.

7.A.4 PROJECT CONTROL SYSTEM: An APSE may provide a tool for keeping track of progress of a project against review dates and budgets. This may include productivity measurements and programmer usage logging.

7.A.5 CONFIGURATION CONTROL SYSTEM: The history attributes provided at the KAPSE level record a variety of software configuration relationships. Tools to help structure these relationships, modify them, indicate the ramifications of (potential) modifications, etc., are appropriate in an APSE. In many systems the facility will be provided, subject to suitable controls, to archive or delete superceded material in the database or to rederive material subsequent to and affected by changes.

7.A.6 MEASUREMENT: Instrumentation and performance measurement tools may be provided by an APSE. These are for use both in determining the efficiency of other tools and in the study of user programs.

7.A.7 FAULT REPORT SYSTEM: As part of the continuing responsibility during maintenance, an APSE may provide a tool to handle error and change reports, to progress them and to relate them to source code documentation and configuration changes. This tool may offer further support for the testing process, for example in automatic retesting after changes.

7.A.8 REQUIREMENT SPECIFICATIONS: An APSE may provide tools for manipulating requirement specifications and for tracing requirement specifications through to design and coding stages.

7.A.9 DESIGN: Tools to aid both system and program design may be part of an APSE.

7.A.10 VERIFICATION: As automatic program verification systems become available, they may be provided in an APSE.

7.A.11 TRANSLATORS: Complex translators may be appropriate in an APSE.

These complex tools may consist of several discrete components that may be interchanged and used by other tools in the APSE. These components may include:

a) recognizer-parser
b) symbol table builder-processor
c) semantic analyzer
d) various analysis and transformation tools for program optimization
e) code generators for various target machines including the host
f) debugging interpreter capable of working with programs in intermediate and mixed host-target object representations.

7.A.12 COMMAND INTERPRETERS: Complex command interpreters may be provided in an APSE. A general command language processor (processing an Ada-based command language, for example) and powerful command editing (from interactive devices) are examples of facilities available in such an APSE tool.

7.B APSE LIBRARIES

7.B.1 It is expected that the standard libraries provided with APSEs will be oriented towards different application areas and/or methodologies as addressed by each APSE.

7.B.2 As one initial example, there will be a requirement for a numeric applications library.

DATE
ILMED
-8